Introduction to Big Data Assignment-7

Name: Faizan Mulla
Roll No: 21F1003885

Problem Statement

1. Create a file with at least 1000 rows of data and upload it to a Google Cloud Storage (GCS) bucket.

2. Write a Kafka Producer application to:
   - Read data from the GCS bucket.
   - Process data in batches of 10 rows and send each batch to a Kafka topic.
   - Introduce a delay of 10 seconds between sending each batch.

3. Write a Spark Streaming Consumer application to:
   - Read data from the Kafka topic every 5 seconds.
   - Compute and emit the count of rows processed in the last 10 seconds.

4. Run the Producer and Consumer simultaneously, stopping after all 1000 rows are processed.

Solution

The solution involves the following key steps:

1. Setting up the environment locally and on GCP.
2. Implementing the Kafka Producer to read from GCS and send data to Kafka.
3. Writing the Spark Streaming Consumer to process Kafka messages.
4. Running and verifying both components simultaneously.

Implementation Details

Step 1: Environment Setup

Local Setup

1. Install required packages:

```bash
sudo apt update; sudo apt -y upgrade
sudo apt install -y python3.11-venv openjdk-11-jdk
```

```bash
python3 -m venv .venv
source .venv/bin/activate
pip install -U kafka google-cloud-storage pyspark pandas
```

2. Download and extract Kafka:

```bash
wget https://downloads.apache.org/kafka/3.8.0/kafka_2.13-3.8.0.tgz
tar -xzf kafka_2.13-3.8.0.tgz
rm kafka_2.13-3.8.0.tgz

cd kafka_2.13-3.8.0/
```

GCP Setup

1. Create a GCS Bucket
   - Navigate to the Google Cloud Console.
   - Create a new bucket named `**iitm-ibd-ga7**`.
   - Upload the 1000-row CSV file to this bucket.

2. Dataproc Cluster Creation

   Created a Dataproc cluster with following specifications:

   - chose the option: Create cluster on compute engine
   - manager node: series → e2 // machine type → e2-standard-2 (2vCPU, 1 core, 8GB memory)
   - reduce primary disk size from 500GB to something less like 50GB.
   - exact same settings for worker nodes too.
   - Region: **asia-south-1**

   - in the customize cluster menu, **uncheck** the INTERNAL IP ONLY option.

Step 2: Kafka Producer Implementation

1. Producer Script (`kafka_producer.py`):

```python
from kafka import KafkaProducer
import pandas as pd
import json
import time
from typing import List


class SalesDataProducer:
    Tabnine | Edit | Test | Explain | Document | Ask
    def __init__(self, bootstrap_servers: str, topic_name: str):
        self.producer = KafkaProducer(
            bootstrap_servers=bootstrap_servers,
            value_serializer=lambda x: json.dumps(x).encode("utf-8"),
        )
        self.topic = topic_name

    Tabnine | Edit | Test | Explain | Document | Ask
    def send_batch(self, records: List[dict]):
        for record in records:
            self.producer.send(self.topic, value=record)
        self.producer.flush()

    Tabnine | Edit | Test | Explain | Document | Ask
    def process_file(self, file_path: str, batch_size: int = 10):
        df = pd.read_csv(file_path)
        records_processed = 0
        total_records = min(1000, len(df))  # Process maximum 1000 records

        while records_processed < total_records:
            batch_end = min(records_processed + batch_size, total_records)
            batch = df.iloc[records_processed:batch_end].to_dict("records")

            print(f"Sending batch of {len(batch)} records...")
            self.send_batch(batch)

            records_processed += len(batch)
            print(f"Total records processed: {records_processed}/{total_records}")

            if records_processed < total_records:
                print("Sleeping for 10 seconds...")
                time.sleep(10)

        self.producer.close()


if __name__ == "__main__":
    producer = SalesDataProducer(
        bootstrap_servers="localhost:9092", topic_name="sales_data"
    )
    producer.process_file("gs://iitm-ibd-ga7/sales_data.csv")
```

Step 3: Spark Consumer Implementation

1. Consumer Script (`spark_consumer.py`):

```
Tabnine | Edit | Test | Explain | Document | Ask
def create_spark_session():
    return (
        SparkSession.builder.appName("SalesDataConsumer")
        .config(
            "spark.jars.packages", "org.apache.spark:spark-sql-kafka-0-10_2.12:3.2.0"
        )
        .getOrCreate()
    )

Tabnine | Edit | Test | Explain | Document | Ask
def create_schema():
    return StructType(
        [
            StructField("timestamp", StringType(), True),
            StructField("user_id", IntegerType(), True),
            StructField("product_id", IntegerType(), True),
            StructField("quantity", IntegerType(), True),
            StructField("price", DoubleType(), True),
        ]
    )

Tabnine | Edit | Test | Explain | Document | Ask
def start_streaming():
    spark = create_spark_session()
    schema = create_schema()

    # Read from Kafka
    df = (
        spark.readStream.format("kafka")
        .option("kafka.bootstrap.servers", "localhost:9092")
        .option("subscribe", "sales_data")
        .option("startingOffsets", "earliest")
        .load()
    )

    # Parse JSON data and convert timestamp to timestamp type
    parsed_df = (
        df.selectExpr("CAST(value AS STRING)")
        .select(from_json(col("value"), schema).alias("data"))
        .select("data.*")
        .withColumn("timestamp", to_timestamp("timestamp", "yyyy-MM-dd HH:mm:ss"))
    )

    # Calculate counts over 10-second windows
    windowed_counts = (
        parsed_df.withWatermark("timestamp", "10 seconds")
        .groupBy(window("timestamp", "10 seconds", "5 seconds"))
        .agg(count("*").alias("record_count"))
    )

    # Start the streaming query
    query = (
        windowed_counts.writeStream.outputMode("update")
        .format("console")
        .option("truncate", "false")
        .trigger(processingTime="5 seconds")
        .start()
    )

    query.awaitTermination()


if __name__ == "__main__":
    start_streaming()
```

Step 4: Execution

In order to open a terminal, go to the running Dataproc Cluster and click on **VM instances**. Now, click on **SSH**, right next to the master node.

1. **Start Kafka (Zookeeper server + Kafka server)**

   Open **Terminal 1**:

   ```bash
   cd kafka_2.13-3.8.0/
   source .venv/bin/activate
   bin/zookeeper-server-start.sh config/zookeeper.properties
   ```

   Open **Terminal 2**:

   ```bash
   cd kafka_2.13-3.8.0/
   source .venv/bin/activate

   bin/kafka-server-start.sh config/server.properties
   ```

   Create Kafka Topic: (*name*=sales_data)

   ```bash
   bin/kafka-topics.sh --create --topic sales_data --bootstrap-server localhost:9092
   --partitions 1 --replication-factor 1
   ```

2. Start **producer service** for `sales_data` topic

   Open **Terminal 3:**

```bash
cd kafka_2.13-3.8.0/
source .venv/bin/activate
bin/kafka-console-producer.sh --topic sales_data --broker-list localhost:9092
```

3. Start **consumer service** for `sales_data` topic

Open **Terminal 4:**

```bash
cd kafka_2.13-3.8.0/
source .venv/bin/activate

bin/kafka-console-consumer.sh --topic sales_data --bootstrap-server localhost:9092
```

4. Run **Spark Consumer**

Open **Terminal 5:**

- Upload the ***spark_consumer.py*** file using “Upload Files”
- Now run this file using:

```bash
source .venv/bin/activate
spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.0 spark_consumer.py
```

5. Run **Kafka Producer**

Open **Terminal 6:**

- Upload the ***kafka_producer.py*** file using “Upload Files”
- Now run this file using:

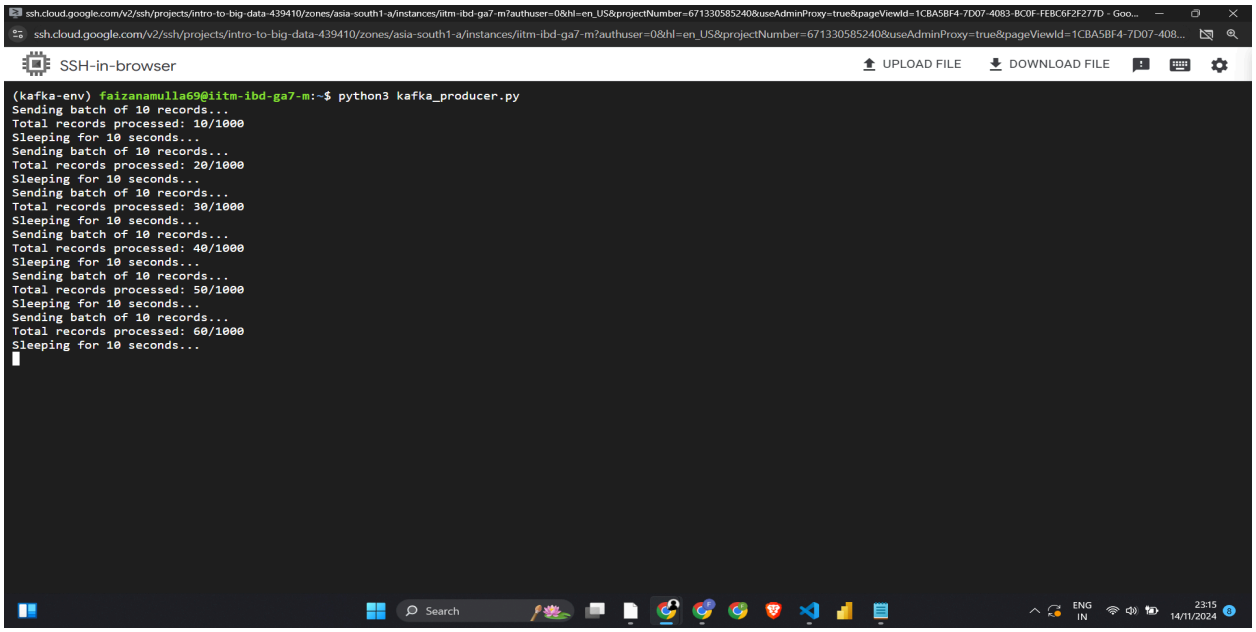
```bash
source .venv/bin/activate
python3 kafka_producer.py
```

<u>Results</u>

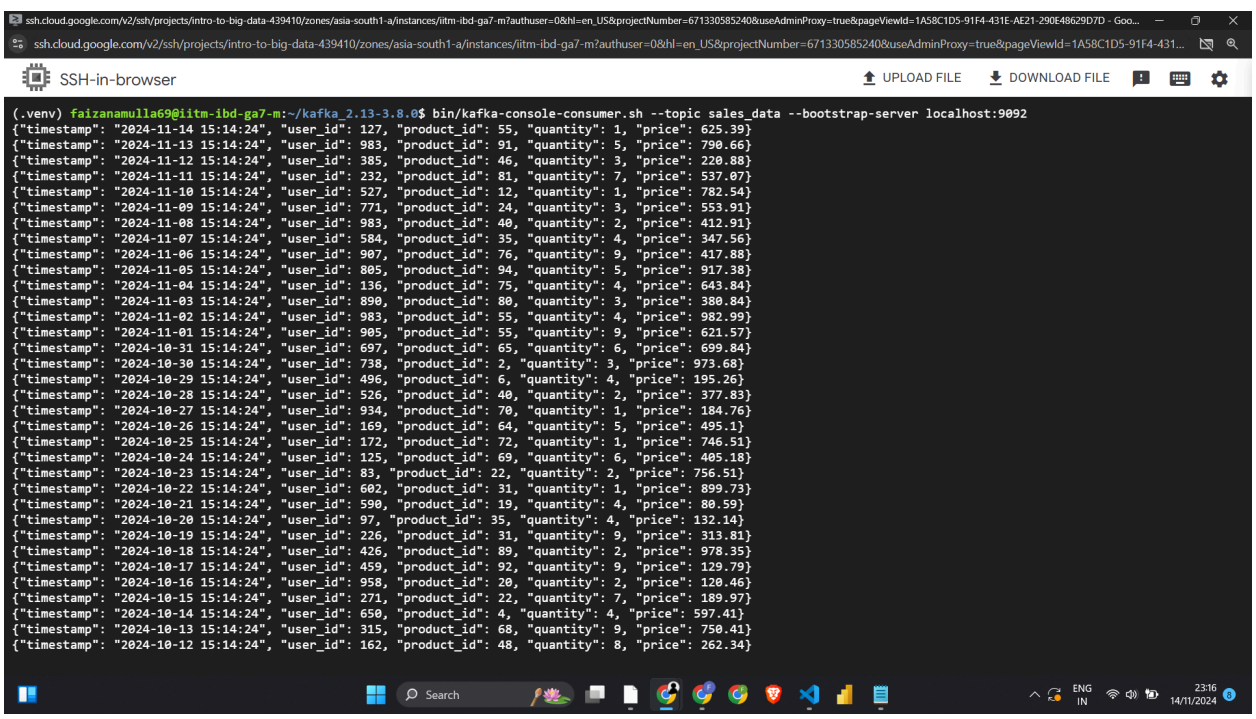
1. Kafka Producer successfully sent 1000 rows in batches of 10 with 10-second intervals.
2. Spark Consumer accurately computed and displayed row counts every 10 seconds.
3. The process stopped after all rows were processed.

Relevant Screenshots

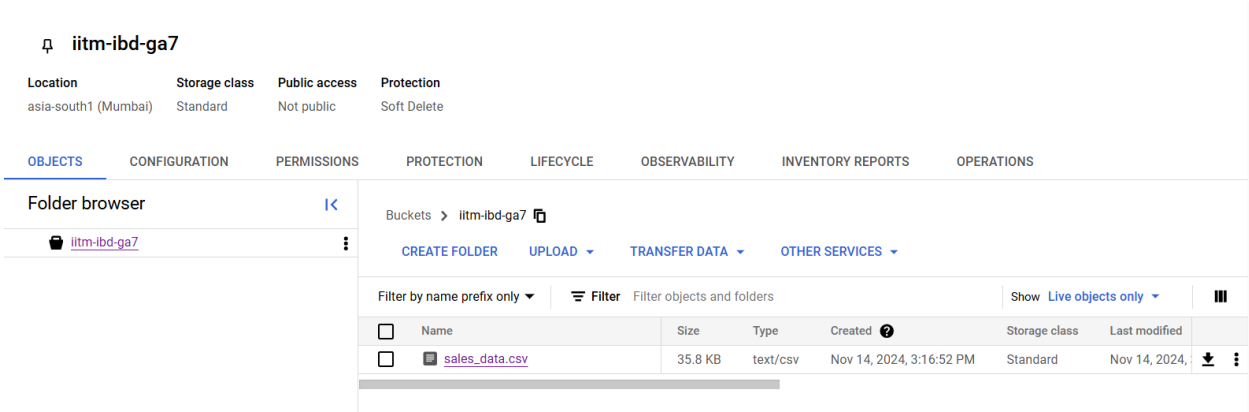
1. Kafka Producer Output:



2. Spark Consumer Output:



3. GCS Bucket:



4. Dataproc Cluster: